LEANDRO BATISTA RIBEIRO

PROF. MARCEL BAUNACH

EMBEDDED AUTOMOTIVE SOFTWARE - VU WS 2016/17

EMBEDDED AUTOMOTIVE SYSTEMS GROUP
INSTITUTE FOR TECHNICAL INFORMATICS
GRAZ UNIVERSITY OF TECHNOLOGY

October 4, 2016

Contents

1 Preface

This lab is about learning and applying some concepts in embedded automotive software development by using the well established OSEK operating system and AUTOSAR middleware standards. By the end of this course, you will have used and understood many operating system constructs in OSEK/AUTOSAR which will help you to create embedded automotive applications and to understand the automatically generated code from high-level design tools. The exercises will mainly be done in plain C on the Infineon TC297 multi-core microcontroller, which will be provided on the Application Kit TC2X7 as depicted in Figure 1.1.

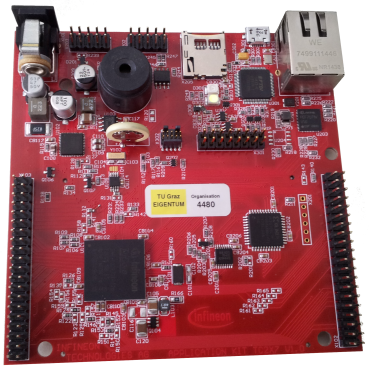


Figure 1.1: The TC297 Application Kit TC2X7.

We want you to be successful! The EAS Lab takes place in conjunction with a lecture, given by Prof. Dr. Marcel Baunach. While the lecture provides theoretical foundations on software concepts in the automotive domain, the lab lets you improve your practical skills through hands-on exercises.

Apart, the lab requires a sound understanding in a broad spectrum of topics *and* the ability to link this knowledge in a well-structured and creative way. Even though there are no formal requirements on previously passed courses, it is highly advisable to bring in some knowledge about "Programming in C" and "Microcontrollers".

How does the lab work? Most exercises consist of tree parts. The first part is the self-studying part, where you should accumulate knowledge by reading some chapters from the provided documentation. With the theory in mind, you will be able to solve the following parts. The second part always consists of a practical exercise that is to be solved or implemented at home, and consequently tested in the lab during the regular course times. Prepare the code well, and make sure that it at least compiles before coming to the lab since you will only have limited time to solve remaining problems using the hardware.[1] In the last part you will have to answer some questions about what you have just learned.

[1] Due to the limited number of development kits and debug equipment, the hardware has to stay in the lab and must not be brought home.

You will be evaluated! The quality of your work will be evaluated continuously. Therefore, the lab is organized in four exercises with self-studying, development tasks, and theoretical questions. For every exercise you will have to hand in the solutions in digital form. We will pay special attention to well-documented source code and to precise

answers to the questions at the end of each exercise. At the end of the EAS Lab you will be evaluated regarding your knowledge about the content of the lecture and the work done in the lab. Details about the evaluation mode will be available in the TU Graz TeachCenter.

A FINAL WORD! Passing the EAS Lab will probably require a significant amount of time. Nevertheless, we believe that the course content will give you a good understanding about the design and implementation of embedded automotive software. On the other hand, the lab is new and this is the second iteration. Thus, some unexpected problems are likely to occur, and the provided documentation or source code will also not be perfect yet. We thus ask for your understanding, and also ask you to help us in improving the course for the iterations ahead. Suggestions are always welcome!

Have Fun and Success!

2 Toolchain Installation

In this chapter we will guide you through the toolchain installation process for Windows. Note that the used compiler is only supported under Windows and will not work as expected in a virtual machine. Please do not try to install anything before reading the rest of this chapter! Some details are important for the correct installation of the toolchain. Before testing the first project, please restart your computer.

2.1 The HighTec Development Platform

The HighTec Development Platform is an Eclipse based IDE developed by HighTec EDV-Systeme GmbH. It supports many silicon platforms like ARM, RH850 (Renesas), PowerPC Architecture, and the TriCore/Aurix platform used in this course. The IDE is used for developing, compiling, and debugging. In particular, it also supports multi-core architectures.

HighTec offers different IDE versions, but for our exercises the free version is sufficient. It can be found under `http://free-entry-toolchain.hightec-rt.com/`. To download the IDE, you have to register yourself for free and the required license will be sent by e-mail (this license does not work in a virtual machine!).

2.2 Java

A Java Runtime Environment (JRE) is required for the Eclipse IDE and for the compiler. Please download the current **32-bit** version, independent of your computer architecture, from `https://www.java.com/en/`.

2.3 Cygwin

Cygwin provides a Unix-like environment under Windows. This is necessary for compiling the operating system, because its make files make use of some Unix commands (e.g., `sed`) which are not natively provided by Windows.

This Unix extension can be downloaded from `https://cygwin.com/install.html`. While versions for 32-bit and 64-bit systems exist, **we recommend to use the 32-bit version.** During installation add the `make` utility (see Figure 2.1).

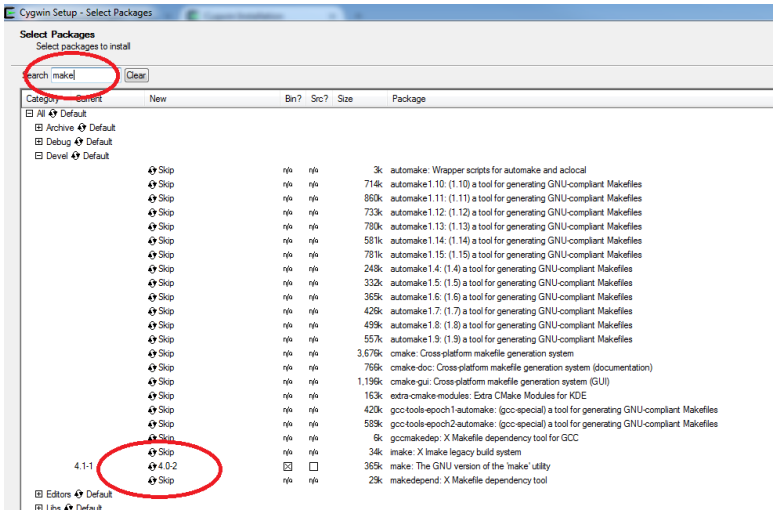


Figure 2.1: Add `make.exe` during installation.

The Cygwin installation also does not automatically add the binary path to the system variables. Therefore, this must be done manually: Click Start → right-click on Computer → Properties → Advanced system settings (on the left) → Environment Variables → Path (under System variables) → Edit. Append the Cygwin bin path to the existing path (by default: "`;C:\cygwin\bin`").

2.4 *Git*

Git is an open source version control system. We will use it to maintain the tool environment, the used base projects, and to upload your exercises. You can use any Git client you prefer. However, we recommend git-scm from `https://git-scm.com/` or TortoiseGit from `https://tortoisegit.org/`.

3 Get ready

After a successful toolchain installation you can download our prepared development environment from our Git server and check if it compiles without problems. We have already created base projects, which include some basic configurations, so you can focus on the implementation. In this chapter, we present how to download the environment step-by-step, using the proposed GIT client `git-scm` [1]. When using other tools, please **make sure that your tool does not change the line endings in the text files**! Further, this chapter shows how to compile the projects and how to debug them.

[1] Simple tutorial: `http://rogerdudler.github.io/git-guide/`

3.1 Project Checkout

Fist, create and navigate to the folder to which the repository should be downloaded. Since our development environment uses Cygwin, it is important that **the destination path does not contain spaces**! Then, right click on the folder and click on `Git Bash` (see Figure 3.1). After clicking, a console will be open.
With the command

```
git clone --no-checkout
https://git.tugraz.at/ci/EAS_environment.git
```

you will clone the repository without downloading the files. This is necessary because we first need to set a configuration for this project. To do that, enter the folder by executing in the console:

```
cd EAS_environment
```

Afterwards, disable the automatic conversion of the line ending with:

```
git config core.autocrlf false
```

Now, the configuration is done and you can download the content:

```
git checkout
```

Follow the same procedure to checkout your personal repository[2], using the URL `https://git.tugraz.at/EAS1617_<SURNAME>_<MATRIKLNUMBER>.git`

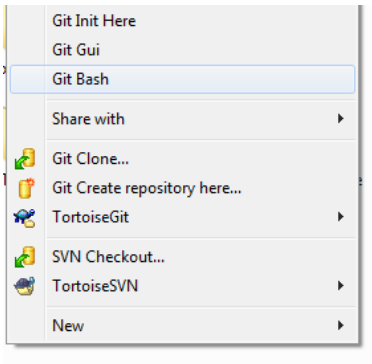


Figure 3.1: How to open git bash.

[2] Instructions for creating and sharing your repository were sent by email

3.2 Project Structure

For better organization, configuration, and maintenance, the project files are located in various directories. Please build your repository

with the same structure as shown in Figure 3.2. **Do NOT change this structure!**

The main working directory is your repository. For each exercise, besides the implementation, create a text-file (`README.txt`) with the answers to the questions.

Figure 3.2 shows further folders from the whole project structure. The `Base_*` directories in the `examples` folder will be used by us to provide a basis for all assignments. The `ee` folder contains the operating system Erika Enterprise. The `RT-Druid` folder includes the tool which generates hardware-dependent C code from the OIL file. Finally, the `utils` folder contains some TC297 configuration files for debugging and flashing the microcontroller.

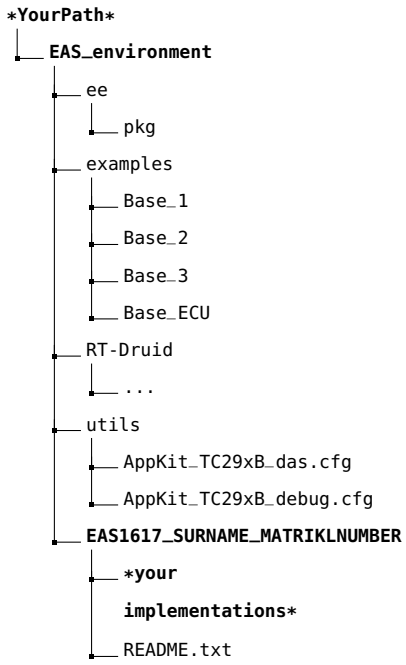


Figure 3.2: Project folder Structure

3.2.1 *How to copy a project?*

Eclipse manages every project in its workspace. In our case, the workspace includes the `Base_*` projects and your individually added projects. For Eclipse, the name of the project not always equals the folder name, but rather the project name is saved inside a configuration file in the project folder. If you simply copy the project folder and import this project into Eclipse you will see the original `Base_*` name. Eclipse shows an error, indicating that the name is used twice. To give the projects a valid name, you have to copy the `Base_*` projects inside Eclipse into your folder.

In Eclipse, copy the `Base_*` project and rename it to the tasks name. Attention! Do not use the default copy directory, instead make sure to save it in your repository folder.

3.2.2 *How to add the files to git?*

To review your assignments, you have to add the project files and to commit and push the commits to TUG server. To eliminate compilation problems, if you are working on different machines, you should not add the `Output` folder to the revision control system Git. To add files to the revision system you can use the git command:

```
git add <files/folders to add>
```

The simplest way to commit all your changes is first cleaning the project[3] and then running the following command from the project folder:

```
git commit -a -m '<description>'
```

The `-a` automatically adds all modified/deleted files to the commit[4] and the `-m '<description>'` uses `'<description>'` as the commit message.

All your commits are stored only locally unless you finally push your changes to the TUG server with the command:

```
git push origin
```

[3] On HighTec, right-click on the project -> Clean Project

[4] Note that you still must manually add new files, like the README.txt, showed on figure 3.2

3.3 Compilation

You have already installed the IDE and checked out the environment. Now, you can check if the base projects compile correctly. To use the base projects in the Eclipse environment, you have to import them first. When starting "Eclipse for Tricore", from the "free AURIX Entry Tool Chain", you will be asked to select the workspace. You should select the `YourPath` folder outside our provided project environment and your repository (see Figure 3.2). After the IDE has started, go to: File → Import → General → Existing Project into Workspace. Browse the examples folder from the project environment, select all projects, and press the `Finish` Button.
Now you can see the imported base projects in the HighTec Project Explorer on the left. To compile a base project, right click on it in the HighTec Project Explorer and click `build`. The console output will show the compilation progress, including warnings and error messages. If the message "*Compilation terminated successfully!*" is shown, there were no compilation errors.

3.4 Debugging

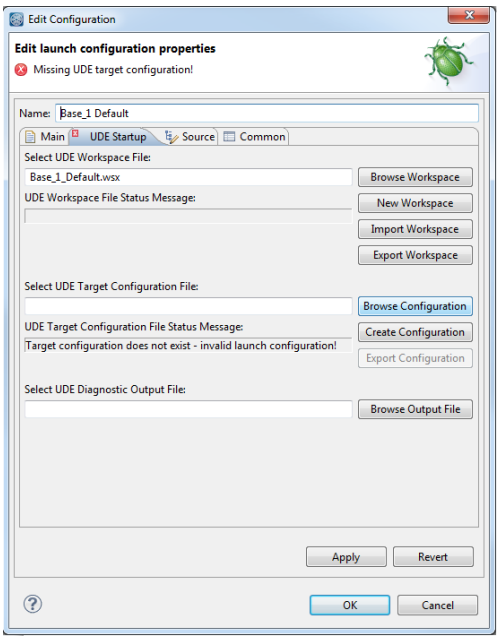
This section shows how to debug the compiled project. The debugger can only be configured if the executable file (`*.elf`) has already been successfully created and the Eclipse IDE has found it. The IDE does not update the file structure in the Project Explorer, therefore you have to do a refresh on the project. Click on the project and press F5, a `Binaries` folder will be shown. If not, close and reopen the project and check again.

For debugging, right click on the project → Debug As → Universal Debug Engine. A configuration window will open (Figure 3.3). Go to `UDE Startup` and Browse the configuration. A pre-made configuration is available in the folder `utils`. Finally, choose the "`Application Kit 2X7 with TC297 B-Step` **`(Multicore Configuration)`**" (see Figure 3.3).

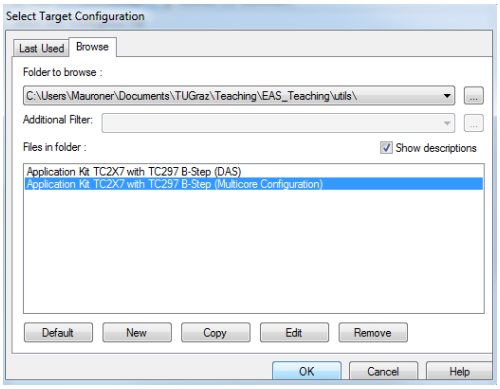
After clicking OK, the error message "Errors in Workspace" can be ignored by clicking `Yes`. You can also tick the corresponding checkbox, so this error message will not show up again.

In case of debugging Problems

The debugger sometimes aborts the download process of the executable file (bug known by the IDE developer). The process then stops with the error message "File System Error" or "Error: Unknown file format" in the `WorkspaceManager - Message View` output, which is a problem of the debugger. One solution is to rebuild the project and then reload the executable file (click on File → Load Programm → press OK). In most cases, this solves the problem. If the error message persists, try to restart Eclipse (not restart inside the IDE!).



(a) Edit Configuration.



(b) Select Target Configuration.

Figure 3.3: Debug configuration windows.

3.5 *Development Kit*

In our Embedded Automotive Software Lab we will use the evaluation board "Application Kit TC2X7" from Infineon, as depicted in Figure 3.4. The board is equipped with an AURIX TriCore™ microcontroller, Infineon's MCU family for the automotive industry, featuring devices with up to 3 cores. On our evaluation board we have the TC297 with three cores and a maximum system clock of 300 MHz.

The Application Kit TC2X7 is equipped with a variety of peripherals for connecting to the environment, like LIN, CAN, Ethernet, and many I/O pins for extension boards or for the mapping of peripheral functions. Moreover, the available touch-display/button will also be used in our exercises along with the four LEDs next to it. Figure 3.4 shows the placement of the touch-display and the LEDs.

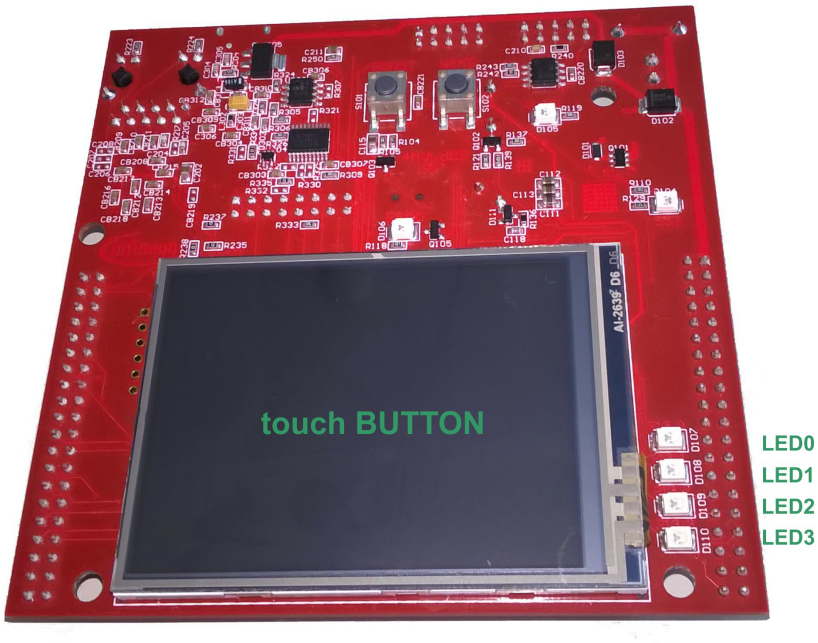


Figure 3.4: Infineon Application Kit TC2X7.

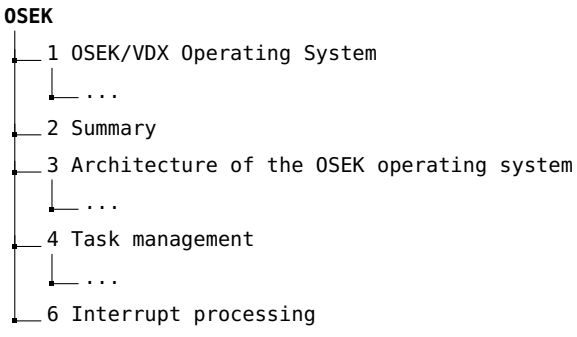
3.6 *Familiarize yourself with the environment*

Meanwhile, you have already installed the IDE and the base project compiles fine. Before you can start with the first exercise, you have to familiarize yourself with the operating system Erika Enterprise and its tools. First you have to become familiar with the OSEK concept, and read some documentation. The aim is to thoroughly understand OSEK and Erika OS which is used for the next exercises. Read the chapters indicated by Figure 3.5(a) from [5] and Figure 3.5(b) from [6].

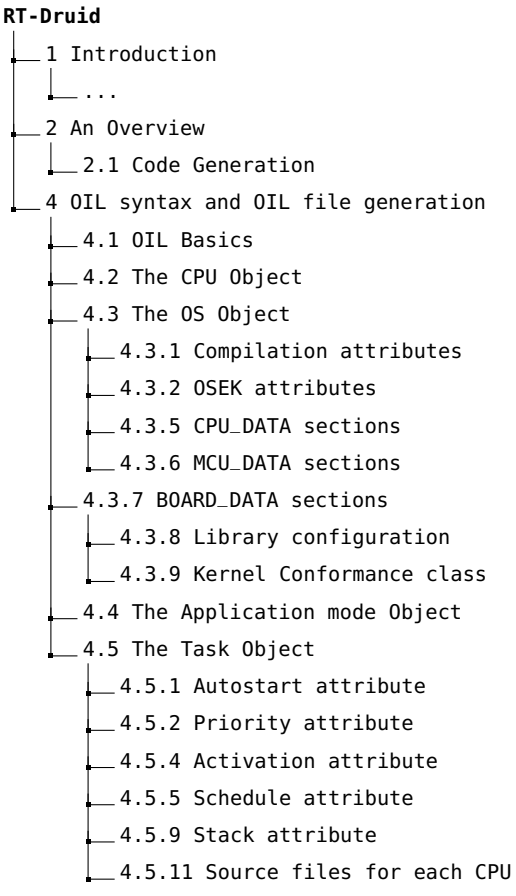
As a quick reference, relevant `OIL` definitions and examples, as well as Erika OS API calls, are provided on the Appendix.

[5] OSEK. OSEK/VDX - Operating System. `http://portal.osek-vdx.org/files/pdf/specs/os223.pdf`, February 2005

[6] Evidence. RT-Druid reference manual. `http://download.tuxfamily.org/erika/webdownload/manuals_pdf/rtdruid_refman_1_5.0.pdf`, November 2012



(a) OSEK Documentation



(b) RT-Druid Documentation

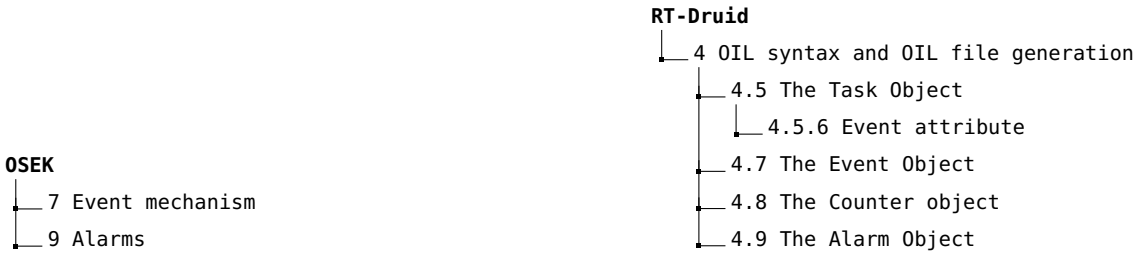
Figure 3.5: Chapters read for introduction

4 Exercise I - Events/Alarms

Abstract. It's time to start! In the meantime you should be familiar with the environment, the OSEK concept and the RT-Druid tool. In this exercise you will use alarms and events. By the end of this exercise, you should have understood how alarms and events work, and how they can be used in an OSEK application.

4.1 Reading

Read the chapters indicated by Figure 4.1 from documents [1] and [2].



(a) OSEK Document

(b) RT-Druid Document

Figure 4.1: Chapters to read about events and alarms

4.2 Event (5 Points)

In this example you should use the touch button to notify a task about an event. The task shall toggle an LED every time an event is received.

To do this exercise we provide the base project named `Base_1`. Copy this project (as shown in Section 3.2.1) and rename it to `assignment_1`. To use the touch button interrupt, you have to define it in the OIL file and initialize it at application start. Additionally, you also have to initialize the LEDs. Figure 4.2 shows the needed C (`.c`) and header (`.h`) files to initialize the components successfully [3]. The implemented interrupt handler should clear the interrupt request and send the event *Event*1 to the task *Task_Event*. In turn, the task shall toggle the LED0 every time it receives the event. The RT-Druid document does not explain how to use the interrupt, but Listing 4.1 shows an example of how do deal with it.

[1] OSEK. OSEK/VDX - Operating System. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf, February 2005

[2] Evidence. RT-Druid reference manual. http://download.tuxfamily.org/erika/webdownload/manuals_pdf/rtdruid_refman_1_5.0.pdf, November 2012

[3] Hint: The interrupt level for the button, during the initialization, must be the same as in the OIL file.

```
ISR handler_name {
  CATEGORY = 2;
  PRIORITY = 2;
};
```

(a) Interrupt Definition in RT-Druid (OIL code)

```
ISR2(handler_name) {
  /* clear interrupt flag */
  /* code */
}
```

(b) Interrupt Definition in Erika Enterprise (C code)

Listing 4.1: Interrupt usage

4.3 *Alarm (5 Points)*

This exercise should toggle an LED periodically by using OSEK Alarms. Continue working on the project `assignment_1`.

Controlled by an alarm Alarm, the task *Task_Alarm* shall start periodically every 500 ms. The sole functionality of the task *Task_Alarm* is to toggle LED1.[4] The task should not start immediately, therefore the phase shift time Φ should be set to 500 ms.

4.4 *Alarm/Event (5 Points)*

In the last exercise of this assignment you should use an alarm to set an event. Still use project `assignment_1`.

Task *Task_Mix* shall toggle LED2 everytime the event *Event_Mix* occurs. This event is toggled with a period of 500 ms by the alarm *Alaram_Mix*.

Notes

Internally, every alarm needs an OSEK counter. For our TC297 use the OIL configuration from Listing 4.3.

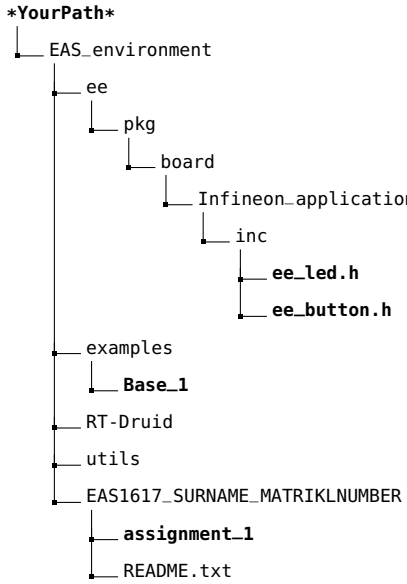


Figure 4.2: Needed files for this assignment.

[4] Hint: Do not forget to terminate the task and to initialize the LEDs.

```
COUNTER xxx_NAME_xxx {
        MINCYCLE = 1;
        MAXALLOWEDVALUE = 2147483647;
        TICKSPERBASE = 1;
        TYPE = HARDWARE {
            DEVICE = "STM_SR0";
            SYSTEM_TIMER = TRUE;
            PRIORITY = 1;
        };
        SECONDSPERTICK = 0.001; // 1ms
    };
```

Listing 4.3: Counter initialization for Infineon AURIX TC297 in the oil file

4.5 *Questions (5 Points)*

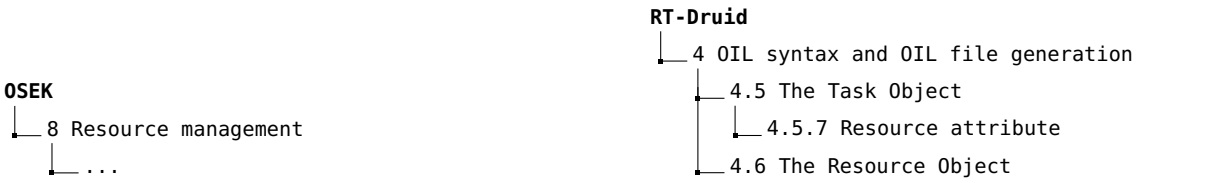
1. How are the priority levels defined for all three processing levels of the OSEK operating system?
2. Explain the difference between the four conformance classes in OSEK.
3. What is the difference between a basic and an extended task?
4. Which scheduling policies exist in OSEK?
5. Explain the difference between category 1 and category 2 interrupts.

5 Exercise II - Resources

Abstract. For most applications, computational resources have to be exclusively used by one task or core. OSEK offers a protection mechanism to prevent race conditions for shared resources on one core at a time. In this exercise, you should learn how to deal with an exclusive resource in OSEK.

5.1 Reading

Read the chapters indicated by Figure 5.1 from [1] and [2].



(a) OSEK Document

(b) RT-Druid Document

Figure 5.1: Chapters to read about resources

5.2 Resources (5 Points)

Use the `Base_2` project as the basis for this assignment and give it the name `assignment_2_1`. On the board's display you will see a black square and a clock which are drawn by different tasks. The two tasks concurrently use the board's display without concurrent access protection, therefore the display will be flickering every time; a so-called race condition occurs over and over. To solve this, you have to protect the concurrent access to the framebuffer (all `draw_*` and `graph_*` calls) and to the display update command (`printScrn`). Make sure that the protection time is minimized! For debugging, LED0 and LED1 are toggled on each task activation.

5.3 Deadlock (5 Points)

Use the `Base_1` project as the basis for this assignment and give it the name `assignment_2_2`. In this exercise you should create **two** tasks and **two** resources which would end up in a deadlock if no deadlock protection exists (Hint: Start both tasks with a different OSEK Alarm).

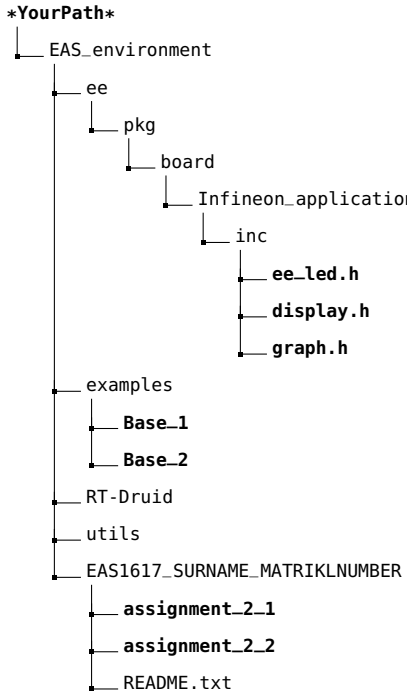


Figure 5.2: Used files for this assignments.

[1] OSEK. OSEK/VDX - Operating System. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf, February 2005

[2] Evidence. RT-Druid reference manual. http://download.tuxfamily.org/erika/webdownload/manuals_pdf/rtdruid_refman_1_5.0.pdf, November 2012

For debugging, you can toggle the LEDs on the demo board to show the state of both tasks.

5.4 *Questions (5 Points)*

1. Which resource management protocol is used in OSEK? Explain it.
2. Give a (simple) example for a priority inversion.
3. Why couldn't you experience a deadlock in the exercise from Section 5.3? Explain how OSEK prevented it.

6 Exercise III - Multi-Core

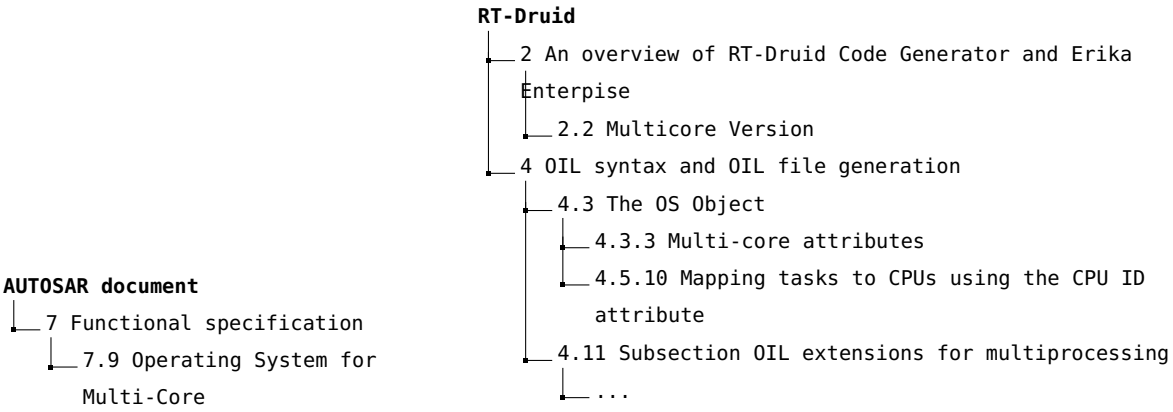
Abstract. In the previous assignment you have used the display from within different tasks and added a protection mechanism to prevent concurrent access on a single core. In this assignment, you should protect the display from concurrent access across cores in a multi-core environment. However, the OSEK standard is designed for single-core systems only. The AUTOSAR standard, which is nowadays used by many automobile manufacturers, adds support for multi-core environments. In this assignment you will learn how to protect resources across cores.

6.1 Reading

Read the chapters indicated by Figure 5.1 from [1] and [2].

[1] AUTOSAR. AUTOSAR - Specification of Operating System. http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf, 2014

[2] Evidence. RT-Druid reference manual. http://download.tuxfamily.org/erika/webdownload/manuals_pdf/rtdruid_refman_1_5.0.pdf, November 2012



(a) AUTOSAR Document

(b) RT-Druid Document

Figure 6.1: Chapters to read about Multi-Core Operation

6.2 Multi-Core (5 Points)

Use the `Base_3` project as the basis for this assignment and rename it to `assignment_3`. Within this exercise and our MCU you have three cores and therefore truly parallel execution. The base project already provides a project with the multi-core start-up code, and each task is assigned to a different core. Two of these three tasks already use the display. Each one draws a different color square,

which is not correctly displayed because of race conditions. Apart, LEDs are also used concurrently by all three tasks. Now you have to implement a protection mechanism across the cores, to protect both resources (display and LEDs) from concurrent access. Make sure that the protection time is minimized and **do not move** the LED and display commands in the source code.

Notes

The RT-Druid document does not explain how to use spinlocks in the oil file. Instead, Listing 6.3 shows the extended definition for the oil file. The simple spinlock configuration method is sufficient for this assignment.

6.3 Questions (5 Points)

1. Which relationship exists between AUTOSAR and OSEK?
2. How does the start-up of a multi-core AUTOSAR system work?
3. Does AUTSAR apply a partial (asymmetric multiprocessing, AMP) or a global (symmetric multiprocessor, SMP) scheduling approach?
4. What is the difference between OSEK resources and the resource management concept used for multi-core architectures in AUTOSAR?

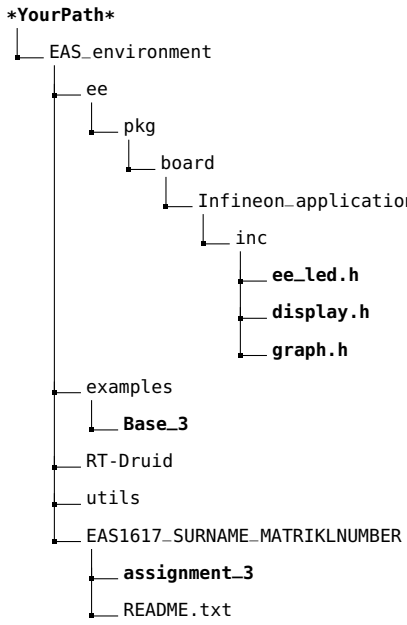


Figure 6.2: Used Files for this Assignment.

```
ENUM [SIMPLE, QUEUED] SPINLOCKS = SIMPLE;

SPINLOCK {
    SPINLOCK_TYPE NEXT_SPINLOCK;
}
```

Listing 6.3: OIL Definition Extension

7 Exercise IV - ECU

Abstract. After a short introduction to the OSEK and AUTOSAR operating systems, you will implement a simple safety-critical ECU by yourself. In this exercise you will have the opportunity to jointly use all previously mentioned OSEK/AUTOSAR features (alarms, events, resources, and spinlocks) in a real multi-core environment.

Use the `Base_ECU` project as the basis for this assignment and rename it to `assignment_ECU`. Read the following sections first to get an overview of the ECU you will implement. Make sure to implement exactly according to the specification from Section 7.2.

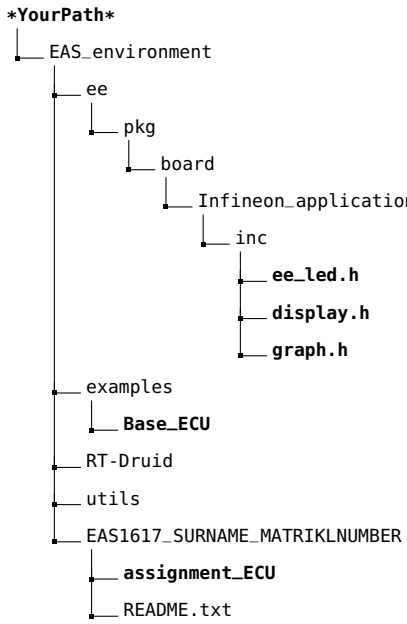


Figure 7.1: Used Files for this Assignment.

7.1 ECU (15 Points)

The goal of this exercise is to implement an ECU which processes and monitors signals from the environment. Therefore, the core c_{slave2} monitors all the output signals of core c_{slave1}. If the output does not correspond to the expected value, the LEDs will indicate an error state. In a real application, the failure would trigger a recovery phase to resolve the problem. In the worst case, a shutdown phase would be triggered to shutdown the ECU in a specific and deterministic way, e.g., if an output signal would not have further effects because of a shortcut in the control line.

In our example, the core c_{master} is used to simulate external input signals. There is a slider on the display to set the simulated ADC value which controls the PWM signal at the output. Furthermore, the display shows a button to simulate an external input signal which activates and deactivates the PWM signal and an additional output-pin. In a real application, these signals would be generated from external analog and digital signals.

7.2 Specification

The task *TaskUI* implements the functionality to simulate the analog and digital input signals. The task's phase shift Φ, which is the initial delay at startup, is 500 ms and its period T is 40 ms. The simulated values are stored into the two global variables `adc` and `on_off`. These

variables are placed into the global RAM memory, where every core has access to them; so they must be protected with a spinlock s. Every change to the `on_off` variable should be notified to the task *TaskPin* via an event *ev*. The task *TaskPin* runs on core c_{slave1}, waits for the event e, and changes the output according to the content of `on_off`. The task *TaskPWM* generates a PWM signal at the output. This task controls the PWM duty cycle which is stored in the `adc` variable and is only generated if the `on_off` variable is set. Otherwise, the output is low. The period T of *TaskPWM* is 1 ms. Its start phase shift Φ is 500 ms. The correct PWM signal generation is more important than the setting of the on-off output signal. Therefore the priority P of *TaskPWM* is higher than the priority of the *TaskPin*. Both tasks use the General Purpose Input/Output (GPIO) component of the MCU. To eliminate potential race conditions, access to the GPIO module (except LEDs) has to be protected by a resource r.

The monitoring tasks *TaskMonitorPin* and *TaskMonitorPWM* both run on core c_{slave2}. *TaskMonitorPWM* has the same task specification as the *TaskPWM*, but the phase shift Φ is postponed by 0.5 ms. With this delay, the task checks the output signal in the half-tick period of the PWM signal. The arrival time a (activation of a non-periodic task) of task *TaskMonitorPin* is 1 ms later than the arrival time of task *TaskPin*. Therefore, *TaskMonitorPin* checks if the realtime requirements are kept. Figure 7.2 shows an overview on all task specifications.

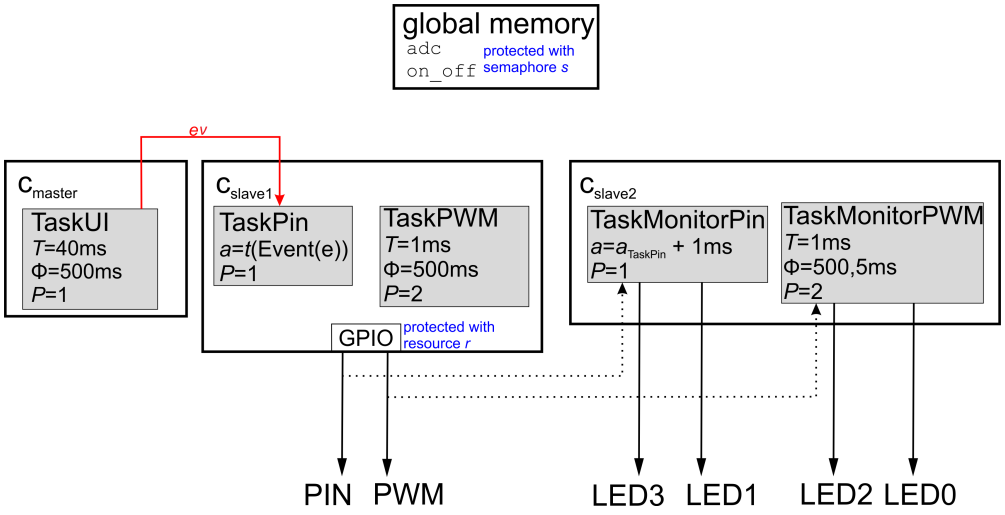


Figure 7.2: Core/Task Structure and Definition.

7.2.1 PIN

This functionality should sense the digital input value (in our case the simulated one) and set the output value to the same value. The monitoring task checks if the output value has been changed within the defined 1 ms deadline. $LED3$ shows the actual output value and, in case the output signal is not correctly set or the output signal was manipulated externally, $LED1$ will turn on to indicate an error.

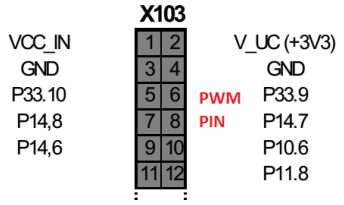


Figure 7.3: Expansion port signal outputs.

7.2.2 PWM

The PWM must exactly follow its specifications, which will be verified by the monitoring task, already implemented on core c_{slave2}. For debugging reasons, the generated PWM signal is also visible on $LED2$. The LED does not show the value of the variable, rather the actual value of the PWM pin.

The PWM has to be implemented as a buffered PWM. This means that the new PWM value is always taken at counter overflow. In our case, the counter runs from 0 to 254 $(0xFE)$ and the new value is updated at counter value 0 (see Figure 7.4). This results in a PWM period of 254 ms. The range between 0 and 254 results in 255 PWM steps, therefore the ADC value range from 0 to 255 corresponds to a duty cycle range between 0 % and 100 %. In case the `on_off` variable is not set, the new PWM value, at counter overflow, will be updated to zero (see Figure 7.4). If the output signal is not set correctly or if it was manipulated externally, $LED0$ will be on to indicate an error.

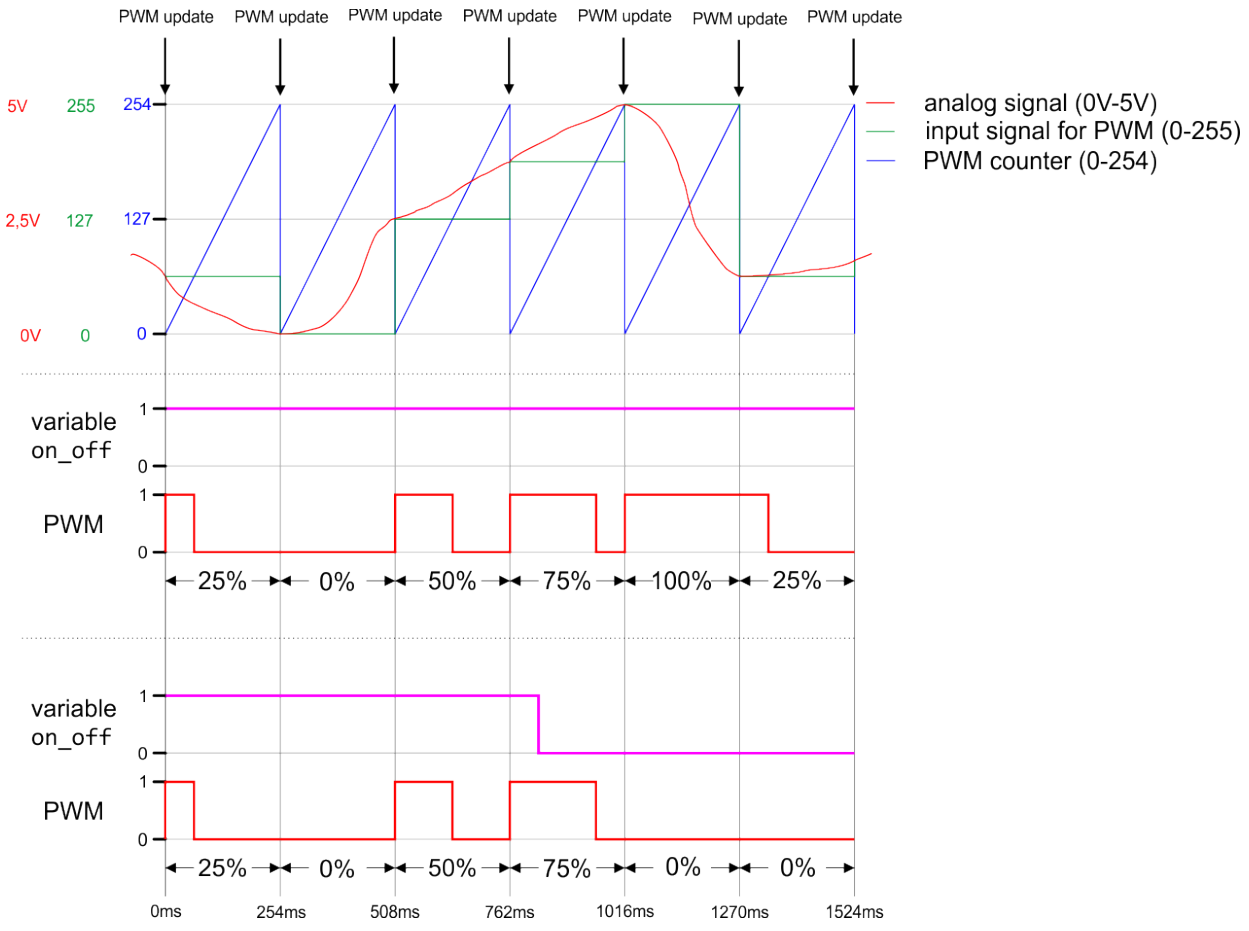


Figure 7.4: PWM Output Examples with different Input Signals.

Appendix

Extended OIL Definition

```
TASK {
  BOOLEAN [
    TRUE,
    FALSE
  ] AUTOSTART;
  UINT32 PRIORITY;
  UINT32 ACTIVATION = 1;     //number of pending activations which can be stored
  ENUM [NON, FULL] SCHEDULE;

  EVENT_TYPE EVENT[];        //events for which this task could wait
  RESOURCE_TYPE RESOURCE[];  //resources used by this task

  ENUM [
    SHARED,
    PRIVATE {
      UINT32 SYS_SIZE;
    }
  ] STACK = SHARED;
  STRING CPU_ID = "default_cpu";  //allocates task to a core
  STRING APP_SRC[];
  ...
};

RESOURCE {
  ENUM [
    STANDARD,
    ...
  ] RESOURCEPROPERTY;
};

ENUM [SIMPLE, QUEUED] SPINLOCKS = SIMPLE;

SPINLOCK {
  SPINLOCK_TYPE NEXT_SPINLOCK;
};


EVENT {
  UINT32 WITH_AUTO MASK = AUTO;
};

ISR handlerName {
  UINT32 CATEGORY;     //1 = internal interrupt, 2 = standard interrupt, 3 = trap
  UINT32 PRIORITY;     //the priority of related interrupt
  ...
};
```

Example

```
TASK taskName {
  AUTOSTART = TRUE;
  PRIORITY = 0x01;
  ACTIVATION = 1;
  SCHEDULE = FULL;

  EVENT = Event1;
  EVENT = Event2;

  STACK = PRIVATE {
    SYS_SIZE = 256;
  };
};
```

```
RESOURCE resourceName {
  RESOURCEPROPERTY = STANDARD;
};
```

```
SPINLOCKS = QUEUED;
```

```
SPINLOCK spinlockName1 {
  NEXT_SPINLOCK=spinlockName2;
};
SPINLOCK spinlockName2 {};
```

```
EVENT eventName  { MASK = AUTO; };
```

```
ISR handlerName {
        CATEGORY = 2;
        PRIORITY = 4;
};
```

```
COUNTER {
  UINT32 MINCYCLE;
  UINT32 MAXALLOWEDVALUE;
  UINT32 TICKSPERBASE;
  ENUM [
    HARDWARE {
      BOOLEAN SYSTEM_TIMER = FALSE;
      UINT32 PRIORITY:                //the priority of related interrupt
      STRING DEVICE;
      STRING HANDLER;
    },
    SOFTWARE
  ] TYPE;
  FLOAT SECONDSPERTICK;               //period
  STRING CPU_ID = "default_cpu";
};
```

```
COUNTER counterName {
  MINCYCLE = 1;
  MAXALLOWEDVALUE = 2147483647;
  TICKSPERBASE = 1;
  TYPE = HARDWARE {
    DEVICE = "STM_SR0";
    SYSTEM_TIMER = TRUE;
    PRIORITY = 1;
  };
  SECONDSPERTICK = 0.001;
};
```

```
ALARM {
  COUNTER_TYPE COUNTER;
  ENUM [
    ACTIVATETASK {
      TASK_TYPE TASK;
    },
    SETEVENT {
      TASK_TYPE TASK;
      EVENT_TYPE EVENT;
    },
    ALARMCALLBACK {
      STRING ALARMCALLBACKNAME;
    }
  ] ACTION;
  BOOLEAN [
    TRUE {
      UINT32 ALARMTIME;
      UINT32 CYCLETIME;
    },
    FALSE
  ] AUTOSTART;
};
```

```
ALARM alarmName {
  COUNTER = counterName;
  ACTION = ACTIVATETASK {
    TASK = Task;
  };
};
```

Used ERIKA OS API calls

```
ISR(Funcname) {...}
TASK(Funcname) {...}

ActivateTask (TaskType TaskID)         //Activates task TaskID, by putting it in the READY state, or in the RUNNING state
TerminateTask (void)                   //Terminates the calling task. After the call, the calling task is set in the
      SUSPENDED state, and it can be reactivated again using ActivateTask, or Alarm notifications
GetResource (ResourceType ResID)
ReleaseResource (ResourceType ResID)

GetSpinlock (SpinlockType SpinlockID)
ReleaseSpinlock (SpinlockType SpinlockID)

SetEvent (TaskType TaskID, EventMaskType Mask) //The call to SetEvent may cause TaskID to wake up from a WaitEvent primitive
ClearEvent (EventMaskType Mask)
WaitEvent (EventMaskType Mask)         //The calling task blocks if none of the events specified in Mask are set. Hint: The
      event is not automatically cleared after leaving the WaitEvent primitive

SetRelAlarm (AlarmType AlarmID, TickType increment, TickType cycle)
```

Bibliography

AUTOSAR. AUTOSAR - Specification of Operating System. http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf, 2014.

Evidence. RT-Druid reference manual. http://download.tuxfamily.org/erika/webdownload/manuals_pdf/rtdruid_refman_1_5.0.pdf, November 2012.

OSEK. OSEK/VDX - Operating System. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf, February 2005.